DS Assignment

Ayush Sahu
11911058
CSE

Date
Page

1) Insertion Sort

In this sort we take array of size n but in for loop we take i from 1 to n.

```
for (i=1 ; i<n ; i++)
{
 for ( j=i ; j>0 ; j--)
 {
    if (a[j] < a[j-1])
        temp = a[j]
        a[j] = a[j-1]
        a[j-1] = temp;
 }
}
```

current num is compared with previous term and swap if req.

⇒ In the worst case inner loop execute x time for every x in order loop.

if x = n.

The no of time loop executes in worst case time $1 + 2 + 3 + \cdots (n-1)$

$$\therefore \frac{n(n-1)}{2} \Rightarrow \frac{n^2 - n}{2}$$

In worst case = $O(n^2)$

So for best case scenario the array is already sorted

And if array is sorted. The inner loop will execute only once

The for every x The inner loop will execute 1 time

Ayush Sahu
1191058

Date
Page

So, time complexity $\Rightarrow (n-1) \times 1$
$= (n-1)$

Time complexity $\leq O(n)$

Example Array size $n = 10$

```
for (i=1 ; i< ; i++)
{
    for (j=i ; j>=0 ; j--)
    {
    }
}
```

Time comp. $= O(9)$ if the array is already sorted the inner loop will run only for 1 time. In the best case.

In the worst case scenario this array is in the decending order and we have to reverse the order (i.e. acending order)

Suggestion

⇒ To reduce complexity of insertion sort in worst case we have to increase it's space complexity

For example array is ~~a[]= {10,9,8,7,6,5}~~

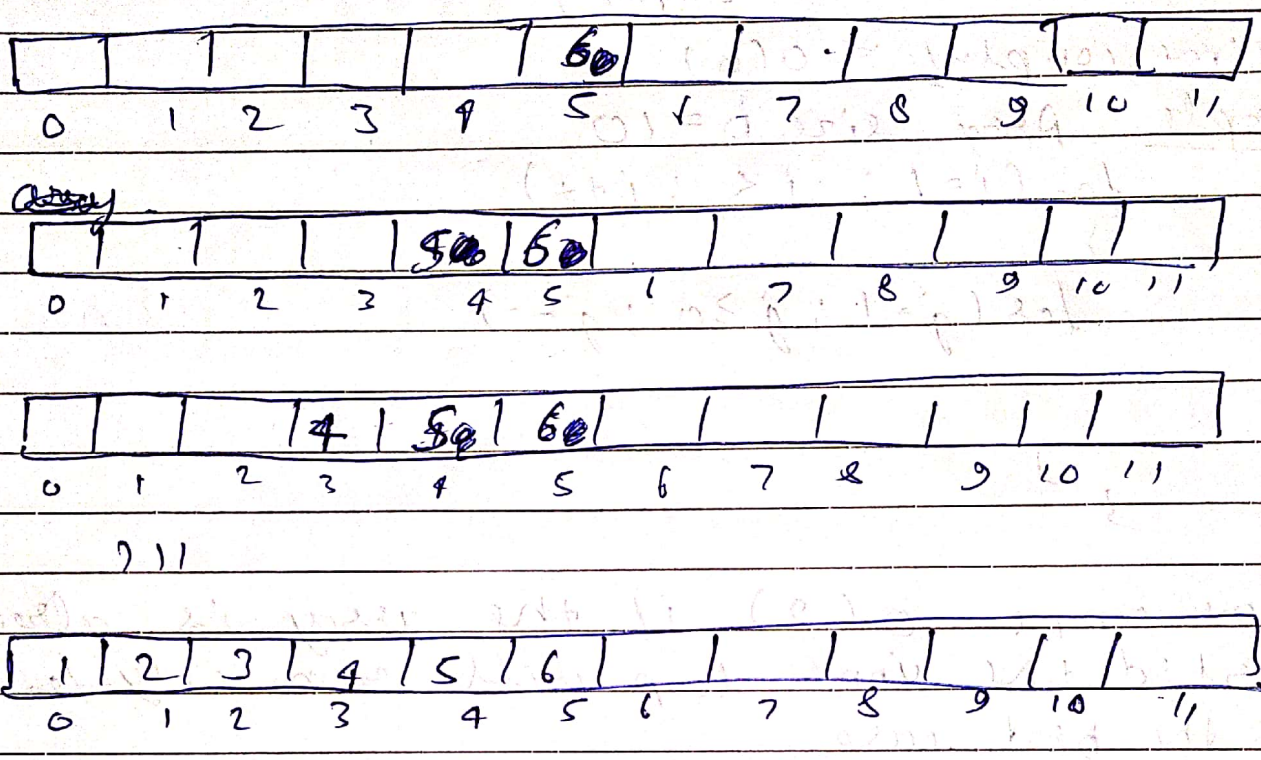
a[ ] = {6, 5, 4, 3, 2, 1}

Using insertion sort to arrange in ascending order

Take array of size 2n if there are n elements to be sorted
and start filling the elements from n+1 position.

Ayush Sahu
1191058

for n=6



So, here the time complexity is O(n) for the worst case scenario.

So, by this time complexity in best and worst case is O(n) by for any other case it is $O(n^2)$

So, hence by these analysis we have reduced the complexity of worst case from $O(n^2)$ to O(n).

Ayush Sahu
11911058

Date
Page

Bubble Sort

It works by repeatedly swapping adjacent element if they are in wrong order. It works the principle the 'light' elmt appears first and heavy at last".

Example we take an array
a: [8, 9, 7, 4, 1]

```
for (i=1, i<n; i++)
  {
    for (j=0, j<(n-i); j++)
    {
      if (a[j] > a[j+1])
      {
        k = a[j]
        a[j] = a[j+1];
        a[j+1] = k;
      }
    }
  }
```

swap if required

3
3

In the first loop

9 > 7 swap
A [8, 7, 9, 4, 1]
9 > 4 swap
A [8, 7, 4, 9, 1]
9 > 1 swap
A [8, 7, 4, 1, 9] → heavy element at last

In second loop

Ayush Sahu
1191c058

A [8, 7, 9, 1, 9]
8 > 7 swap
A [7, 8, 9, 1, 9]
8 > 9 swap
A [7, 9, 8, 1, 9]

⇒ 8 > 1 swap
A [7, 9, 1, 8, 9]

Similarly we get after 1st loop.

A [1, 9, 7, 8, 9]
sorted

Time complexity:
T(n) = O(n-1) × O(n-1)
= O(n²)

~~loop 1 & loop 2~~ operate (n-1) times
loop 2 (n-1) times

T(n) ≈ O(n²)

But cant where all are sorted.

Inner loop will run const time
Outer loop will run (n-1) times.

So T(n) = O(n).

Insertion sort

Size of array is N
Pseudo code

Ayush Sahu
1911058

```
for (i=1; i<n; i++)
{
  temp = a[i]
  j = i-1
  while (j>=0 && a[j] > temp)
  {
    a[j+1] = a[j];
    j--;
  }
  a[j+1] = temp;
}
```

for the Worst case
Inner loop will run (i) times for every i in out loop

Worst case $1+2+3+\dots+(n-1)$

$$\frac{n(n-1)}{2} = \frac{n^2-n}{2}$$

$T(n) = O(n^2)$

Best case Outer loop run for n-1 times and inner loop will run for a constant

So

$T(n) = O(n)$

Merge sort (out place Algo).

```
Merge(A, p, q, s)
{
  n1 = q-p+1
  n2 = s-q
  L[1 ... n1+1] & R[1 ... n2+1]
```

Ayush Sahu
1191058

Inplace Algo

Quick Sort

Any of the point is chosen as the pivot point in this all the value from the pivot point left are less than the value of pivot pt and all the values greater than pivot pt are on the right of it.

Through the point selected (ie pivot) we divide the arr into 2 parts.

Worst Case :- When pivot elmnt is smallest as largest or all elemts are same.
The returned sublist will be of size (n-1)

```
partition (l, m, array)
    int start = l
    int end = m
    pivot = a[i]

while (start < end)
{
  while (a[start] <= piv) && start < m)
    {
    start++;
    }
while ( a[en] > pivot) && (end >= l)
    end--;
  }
  if (start < end)
  {
```

Ayush Sahu
11911058

swap (array[start], array[end])
}
}
swap (array[i], array[end])
return end;

If this happens in every time then the ith call while do O(n-i) work to do the partition

$$\sum_{i=0}^{n} O(n-i) = n + (n-1) + (n-2) + \cdots + 2 + 1$$
$$\Rightarrow \left(\frac{n^2+n}{2}\right)$$
$$= O\left(\frac{n^2+n}{2}\right)$$
$$= O(n^2)$$

Best case: In most balanced case each time we perform a partition we divide the list in two nearly equal pieces. This means each recursive call, process a list of half the size. Consequently we can make only $\log_2 n$ nested calls before we reach a list of size.

So by the we have seen that for the best case time complexity is $O(\log_2 n)$ and for the worst case it is $O(n^2)$.

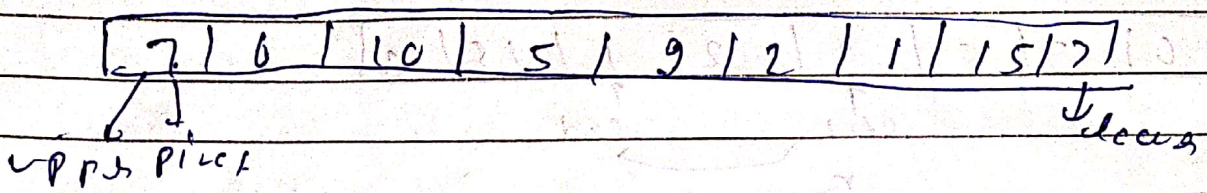
Each level of calls need only $O(n)$ time all together

$$O(n \times \log_2 n)$$
$$= O(n \log_2 n)$$

Ayush Sahu
1911058

Date
Page

Example

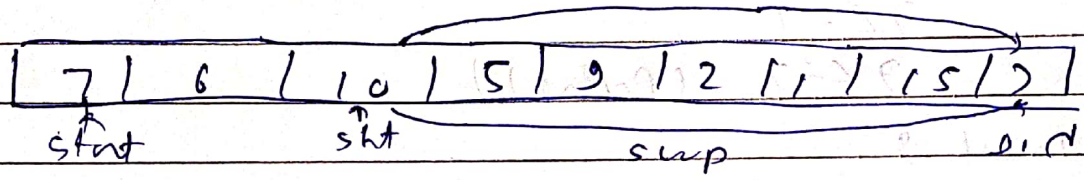


We take first element as pivot and compare it with 6.

6 >= 7 (True)

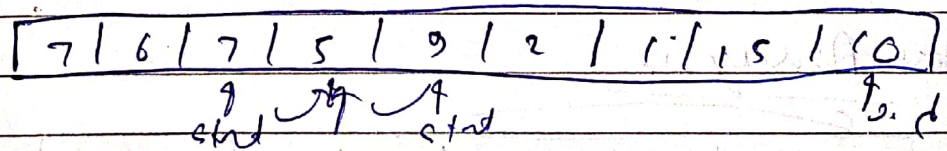
Then compare 10

10 <= 7 (False)



Now we compare from the end any element which is less than eq to 7 and swap with 10

Now array



Now compare

7 <= 10 (Increment start)

5 <= 7 (Start increment)

9 <= 7 (False)

Start will be at 9

Now compare end

7 <= 10

7 <= 15

7 <= 1 (False)

Ayush Sahu
1191058

end will be at 1

| 7 | 6 | 7 | 5 | 9 | 2 | 1 | 15 | 10 |

start → end

Now array

| 7 | 6 | 7 | 5 | 1 | 2 | 9 | 15 | 10 |

start → end

Comparing

1 <= 7
2 <= 7
9 <= 7 (false)

Start at '9'

compare

> <= 9
7 <= 9 (False)

So end at 2

Array is

| 7 | 6 | 7 | 5 | 1 | 2 | 9 | 15 | 10 |

end start

Now start has crossed end so we will not swap instead we will swap end with pivot element at a[0]

Now '7' is at correct position. The same procedure will be followed by taking pivot b/w 2 and 1 & 9 and 10.

| 2 | 6 | 7 | 5 | 1 | 7 | 9 | 15 | 10 |

correct pos

Ayush Sahu
11911058

Date
Page

Merge sort (out place)

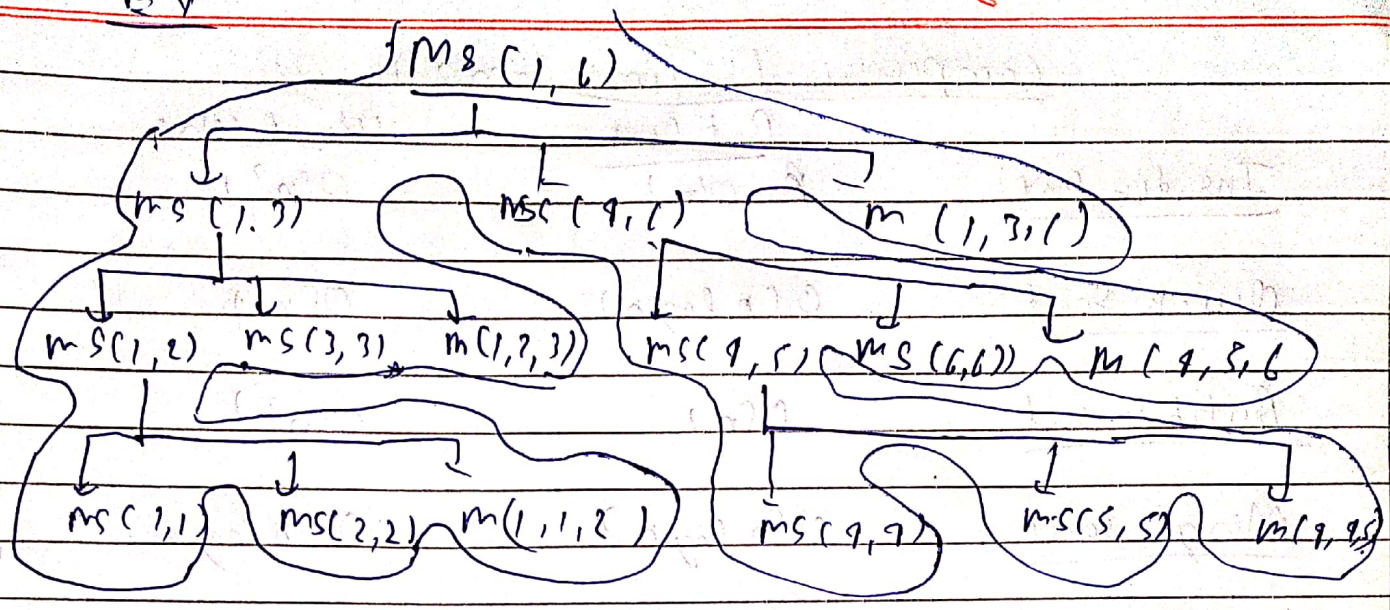
```
Merge (A, p, q, r)
{ n1 = q - p + 1
  n2 = r - q
// L[1..n1+1] & R[1..n2+1]
  for (i = 1 to n1)
  {
      L[i] = A[p + i - 1]
  for (j = 1 to n2)
  {R[j] = A[q + j]}
  L[n1+1] = ∞
  R[n2+1] = ∞
  i = 1  j = 1
  for (k = p to r)
    if (L[i] ≤ R[j])
      { A[k] = L[i]
        i = i + 1}
    else [ A[k] = R[j]
           j = j + 1 } }
```

Merge sort

```
{ if (p < r)
  { q = [(p + r)/2]
  merge sort (a, p, q);
  merge sort (a, q+1, r);
  merge (A, p, q, r)
  }
}
```

Ayush Sahu
1191058

By



Time complexity As it needs for n element for copying taking comp n

and merging take 2 element together so it take $\log_2 n$

$T(n) \simeq O(n \log_2 n)$

In place algorithm are those that does not need an extra space and produces an output in the same memory that contains the data by transforming the input

So quick sort and bubble are implace while merge sort is not place as take space order of O(n)

Ayush Sahu
1191058

Date
Page

Comparison of Time Complexity

	Best Case	Worst case
Insertion Sort	$\Omega(n)$	$O(n^2)$
Quick Sort	$O(n \log n)$	$O(n^2)$
Bubble sort	$O(n)$	$O(n^2)$
Merge sort	$O(n \log n)$	$O(n \log n)$

3) In This program of sorting the string in alphabetical order we first take the string in which we put characters one by one by the user. and use the merge sort algorithm for planning the task In this we have used strcmp( ) for comparing the two characters and if it is in ascending order we that fine otherwise we use strncpy( ) to copy the characters at its require or original position, So by this whole process we have succeded in our task.

4) In this program of making bst tree in the dictionary order first we have taken the strings stored in array and on to make the bst we first create the root and we used the function strcmp( ) - for company the adjacent strings If the value

Ayush Sahu
11911058

Date
Page

is <0 then move the string to the left and carry on and if it is >0 then add the string to the right of the node and we have used strncpy() to copy the string at desired position.